13. Global Features of the VPLX Language

13.1 Introduction

Since the step is the fundamental unit of organization for the VPLX language, most of the documentation mirrors this structure. This chapter, however, discusses two topics that cut across the steps.

The first topic concerns the specification of files. Each step begins with the implied or explicit definition of input and output files for the step. In addition, some steps may reference additional files. The next three sections, 13.2 - 13.4, systematically discuss issues of file specification.

The second is the group of global statements in the VPLX language that may be used anywhere. Contrary to the generally strict organization of a VPLX program into a sequence of steps, the effect of these statements may span steps or appear within any step (*1*). For example, the ECHO command may be used to control the echoing of VPLX commands to the print file, and it takes effect immediately and persists until the next ECHO statement.

Features such as SET, INCLUDE, MACROWRITE, and others provide the VPLX language with macro-like capabilities, permitting the design of relatively complex production systems. The chapter suggests some of the numerous applications of these features.

INCLUDE and MACROWRITE both specify files. In the case of INCLUDE, syntactic restrictions (Sections 13.5.1 and 13.6) are required to avoid conflicts with file specification conventions. Consequently, it is useful to group discussion of file specification and global statements into the same chapter.

This chapter should be consulted when file specification appears problematic or before the design of production systems that may require repetitive VPLX runs. Some of the features, like COMMENT, LONGCOMMENT, ECHO and INCLUDE, are potentially useful in almost any VPLX application.

Sections of this chapter concerning file specification:

- Section 13.2 describes general aspects of file specification.

- Section 13.3 documents system-specific features of file specification.

- Section 13.4 discusses scratch files.

Sections of this chapter describing statements that can be used anywhere:

- Section 13.5 describes INCLUDE, to initiate reading of VPLX commands from another file.

- Section 13.6 documents COMMENT, which is amply illustrated elsewhere in the documentation. The section notes specific interactions between COMMENT and INCLUDE.

- LONGCOMMENT, in Section 13.7, is a variant of COMMENT. Since LONGCOMMENT treats most lines as comments, including those beginning in position 1, it may be used to suspend execution of one or more VPLX commands.

- Section 13.8 describes IGNORE, which directs VPLX to ignore one or more commands that follow. This command is similar to LONGCOMMENT, but it also turns off echoing of statements to the print file.

- ECHO OFF in Section 13.9 may be used to suspend listing of VPLX commands and other information to the print file.

- Section 13.10 introduces SET, which defines substitution strings that may be referenced subsequently in a VPLX command file. This macro-like feature supports the design and control of relatively complex VPLX applications.

- MACROWRITE in Section 13.11 is especially designed to work with the SET feature, but it may be used to create small files directly from the VPLX command file.

- ON and OFF in Section 13.12 are synonyms for COMMENT and IGNORE, respectively. (ON is actually equivalent to a 1-line version of COMMENT; subsequent lines are not treated as comments.) Although neither ON nor OFF has an obvious direct use, they offer an approach, when combined with SET, to organize blocks of VPLX code that may be turned on or off as part of the design of a production system.

- Section 13.13 illustrates many of these features.

13.2 Specifying Files

13.2.1 General form of file specification. VPLX anticipates file specifications in the following instances:

1) At the beginning of steps - CREATE, DISPLAY, *etc.*, may specify one or more files, whose functions are identified by the key words IN=, OUT=, VPLXIN=, *etc.*

2) As part of some subroutines in the TRANSFORM step - REPWRITE, REPREAD, *etc.*

3) As explicitly identified scratch files, SCRATCH1, SCRATCH2, *etc.*

4) As a command file specified by INCLUDE.

5) As a target file for MACROWRITE.

Except for the use of key words at the beginning of steps in 1), the same rules apply to file specifications in all instances.

Files must not be open for two different purposes at the same time, that is, in the same step. For example, no file can be both the input and output of the same step. VPLX will terminate in error when asked to use a file for more than one purpose simultaneously.

File names are limited to 80 characters (*2*). The usual form of file specification is to give simply the name of the file, which is illustrated by the many examples in this documentation. The usual form may be modified in two ways: 1) the file name may be enclosed in apostrophes (single quotation marks), and 2) optional file information inside parentheses may follow.

13.2.2 Use of Apostrophes. VPLX removes single apostrophes at the beginning and end of file names, instead of treating them as part of the name. Thus, apostrophes may be used to delineate complex file names. Two situations require apostrophes:

1) If a blank is part of the file name. For example, to read a file across nodes in VMS, `'dsvx01"name password"::temp$:survey.dat'`

2) If the name includes a left parenthesis - " (" For example, a member of a partitioned data set in IBM OS: `'file1.partitioned.dataset(survey)'`

Without apostrophes, VPLX assumes that the file name begins with the first nonblank character and ends with the first blank. The second situation also requires apostrophes because VPLX otherwise assumes that the " ("marks the beginning of optional file information.

13.2.3 Optional file information. As an example of optional file information, VMS on VAX systems requires that FORTRAN specify a maximum length for long output records, discussed in more detail in Section 13.3. If particularly long records were desired as the output from REWEIGHT, in Chapter 14:

```
REWEIGHT   IN = data1:[000000]survey.dat
  VPLXIN = temp$:repfact.vpl
  OUT  = temp$:repweights.dat (lrecl = 1100)
...
OUTFORMAT (f4.0,f7.0,f3.0,f4.0,101f10.2)
OUTPUT  id1 - id4 weight repw1 - repw100
```

would warn VMS about the long records. Note that the value given did not need to be exact but only a suitable upper bound.

The implementation of VPLX on some systems, such as PC DOS/Windows and UNIX, does not have any additional options available. Nonetheless, a few syntactic restrictions arising from the potential for " (" to follow a file name affect all systems and are described in Section 13.5.

13.3 System-Specific Optional File Information

13.3.1 General Remarks. In some cases, FORTRAN implementations add nonstandard features to the language to achieve a better fit to the operating system. In some cases, commented lines in the VPLX program identify code designed to run under only one system. For example, under VAX VMS, files opened for reading include the specification 'READONLY' in the OPEN statement, thus permitting access to files to which the user has read but not write access.

In some cases, nonstandard choices are also available, and the most useful of these have been incorporated into the VPLX language. As noted in Section 13.2, no extensions are available in the
DOS/Windows version and the UNIX version.

13.3.2 VMS (VAX/Digital) - If a formatted output file includes records that will be longer than 132 characters, then the length of the longest record, in characters, should be specified. Section 13.2.3 provides an example. Key words LRECL or RECL may be used interchangeably to specify the maximum number of characters to be written.

13.3.3 IBM OS, CMS Adaptation of VPLX to the IBM mainframe environment has not been completed, but will use the optional file information for items such as VOL=SER, BLKSIZE, *etc.*

13.4 Scratch Files: SCRATCH1, *etc.*

The primary VPLX steps employ temporary scratch files to varying degrees. Generally, CREATE requires the largest number of files of all the steps: under some conditions, 5 scratch files, although more typically 1 or 2. In general, VPLX considers these files to be SCRATCH1, SCRATCH2, ... SCRATCH5. VPLX steps are written to select SCRATCH1 as the largest anticipated file, SCRATCH2, as the next largest file, *etc.* If the user does not explicitly define the scratch files, VPLX will obtain them under default options for the operating system.

Often, the FORTRAN implementation assigns these scratch files to the user's default directory. If one has a single hard drive C: on a PC, then the system defaults work well. On larger systems, however, it is often advantageous to obtain the storage from another disk partition or source. To do so, the form of the statement is:

```
SCRATCH1   fname
```

with analogous forms for SCRATCH2, SCRATCH3, SCRATCH4, and SCRATCH5. The user may adopt the strategy of assigning only the largest files, say SCRATCH1 and SCRATCH2, and rely on VPLX to obtain any remaining files, which are likely to be small, under the default option.

As in general, VPLX does not enforce any convention on the naming of temporary files. It is recommended, for the purposes of identification, to include some indication of the temporary nature of the files, such as an extension `.tmp` or incorporating TEMP into the name.

Once a scratch file has been defined through SCRATCH1, *etc.*, VPLX will continue to use it, as needed, in successive steps in the same run, unless the file is deleted through DELETE or FREE.

Disposition at End of Run. After normal termination of a VPLX run, VPLX will attempt to delete all scratch files, even those named through SCRATCH1, *etc*. If the program terminates abnormally, however, some scratch files may remain, regardless of whether they were originally named through SCRATCH1, or assigned by default. Should the user discover such left-over files after an abnormal termination, it is best to delete them to recover the space.

13.5 INCLUDE

13.5.1 General Description of INCLUDE: Section 3.4 presented an example of INCLUDE,

```
constants
include    factors1.dat
                into f1 - f20
```

INCLUDE directs VPLX to read from another file as the command file. The syntax is:

```
   include     filename
```

Normally, VPLX continues reading commands from the new file until it reaches the end of file. At that point, control then returns to the file in which the INCLUDE statement appeared.

NOTE: VPLX only reads positions 1-80 from any INCLUDEd file. Thus, if INCLUDE is used with CONSTANTS to introduce data from another file into a CREATE or TRANSFORM step, the numbers must fall within positions 1-80. This restriction does not apply to other types of character input to VPLX governed by formats, e.g., the IN= file to the CREATE step, REPREAD in the TRANSFORM step, *etc*.

NOTE: INCLUDE is the only exception to the general rule that a multi-line command ends when a new command begins with a letter or underscore, " _ ", in position 1. The example at the beginning of this subsection, 13.5.1, illustrates how the material from an INCLUDEd file may be integrated into a single command. Any other command following CONSTANTS in the example would end the scope of the CONSTANTS command, leaving it incomplete. The INCLUDE statement does not end the scope of the CONSTANTS statement.

Nesting of INCLUDE Statements: INCLUDE may point to a file that itself has one or more INCLUDE statements. The process becomes analogous to a stack of plates. Each INCLUDE statement adds another plate to the stack, which is removed each time VPLX finishes reading the file to the end. VPLX is sensitive to the depth of the stack: an INCLUDE in the initial file may point to a first alternate file, which may point to a second alternate file, which finally may point to a third. The stack cannot go deeper than 3 without internal changes to VPLX. It is possible, however, to keep adding plates to the stack as long as plates are also being removed (by reading the end of file) in such a way as to never allow the stack to go too deep.

```
  include  file1.crd
  VPLX now starts reading from file1.crd:
  ..  comment  beginning of file1.crd
  ..  include  file2.crd
  ..   VPLX now starts reading from file2.crd:
  ....  comment  beginning of file2.crd
  ....  include  file3.crd
  ....  VPLX now starts reading from  file3.crd:
  ......  comment  beginning of file3.crd
  ......     Because this file is at the maximum
  ......     depth of the stack, it may not contain
  ......     include statements.
```

```
......  comment  end of file3.crd
....  VPLX returns to reading from  file2.crd:
....  comment  file2.crd may contain additional
....        include statements, if desired
....  comment  end of file2.crd
..  VPLX returns to reading from file1.crd:
..  comment  file1.crd may contain additional
..        include statements, if desired

..  comment  end of file1.crd
VPLX returns to reading from the initial command file:
comment  The main file may contain additional include
         statements, if desired.
```

Exhibit 13.1 Illustration of the maximum depth of include commands. The indentation with ".." attempts to suggest the reading of commands from different files.

13.5.2 Restrictions on INCLUDE: Generally, INCLUDE may point to a file with a series of VPLX commands, to a file with a single command, or even a file with some of the lines of a multi-line command. During multiline substitutions, however, there are two important limitations on INCLUDE. The restrictions are to avoid ambiguity and overly complex program logic within VPLX itself.

One restriction is that, once a file specification is started, INCLUDE must not be used to provide the remaining part of the specification. This is particularly at issue for the initial commands of steps. For example, the following approach is **not allowed**:

```
reweight   in =  testdata1.dat
include    outfname.crd
```

where the file `outfname.crd` was

```
     out = rewt.dat
```

An INCLUDEd file may contain the entire initial statement, however. The following 2-line file may be INCLUDEd:

```
reweight   in =  testdata1.dat
           out = rewt.dat
```

In general, INCLUDE is designed for larger sections of code than a file name. The SET feature, in Section 13.10, is better suited for file specifications and other short sections of VPLX code (*3*).

The second restriction is that, if INCLUDE is used to bring in comments from another file, the next line must not contain "(" as the first nonblank character. Section 13.6 describes this restriction.

13.5.3 Uses of INCLUDE: INCLUDE is quite useful in a wide variety of contexts. To suggest just some of them:

- As in the CONSTANTS illustration above, one may want to INCLUDE values calculated by another VPLX step or another system into the VPLX step.
- If an input data file has many variables, it may be useful to write an INPUT and FORMAT statement to another file and to INCLUDE the file each time it is needed. This approach assures consistence of the information from one step to the next.
- Particularly in the TRANSFORM step, one may want to repeat the same set of commands several times, for example, in carrying out an iterative calculation. The group of repeated statements may be placed in a file and the file INCLUDEd as many times as necessary.
- If a calculation involves a significant number of VPLX steps, it may be useful to divide the steps into separate files and create a master control file that INCLUDEs them. If some steps succeed and not others, one can simply change what files are INCLUDEd from the master control file and restart the run taking advantage of the successful steps.

13.6 COMMENT

The simple rules of COMMENT should be evident from the numerous examples throughout the documentation: COMMENT begins a single or multi-line series of comments, where each continuation line avoids a letter or underscore " _ " in position 1.

It is possible to INCLUDE comments from another file as part of a series of COMMENTS. If this is done, however, users must avoid the restriction detailed in Section 13.5.1, i.e., the first nonblank character on the line following the INCLUDE statement must not be a " (". For example, the combination:

```
comment
include    comments.txt
          (above comments generated by another program)
```

results in an error as VPLX attempts to interpret the material after " (" as optional file information.

13.7 LONGCOMMENT

LONGCOMMENT serves a function similar to COMMENT. Except for INCLUDE, the scope of COMMENT ends with the first letter or underscore " _ " in position 1. As a general rule, the scope of LONGCOMMENT ends only when VPLX encounters a statement LONGCOMMENT OFF.

```
longcomment
statements/text etc. treated as comments
longcomment  off
```

Until the effect of the LONGCOMMENT is turned off, all lines, including those that are valid VPLX commands, are treated as comments, including INCLUDE statements themselves.

The format of LONGCOMMENT OFF is one of the few exceptions to the general rule that VPLX commands may be continued on multiple lines. The "OFF" must appear on the same line as "LONGCOMMENT."

There are two refinements to the preceding rules:

- If LONGCOMMENT is encountered within an INCLUDEd file, its effect is turned off at the end of the file, if a LONGCOMMENT OFF statement has not already done so.

- If a second LONGCOMMENT is encountered after a first, then VPLX waits until two LONGCOMMENT OFF statements have been encountered to resume interpreting the input as commands, unless the end of file is reached first. In general, VPLX waits until the number of LONGCOMMENT OFF statements have balanced off the number of LONGCOMMENT statements before resuming execution. In other words, each LONGCOMMENT must be balanced by its own LONGCOMMENT OFF (*4*).

13.8 IGNORE

IGNORE is similar to LONGCOMMENT in Section 13.7. Both suspend execution of any statements. LONGCOMMENT prints the statements, but IGNORE suspends printing until a matching IGNORE OFF statement is encountered. IGNORE has no effect, however, if it is within the scope of LONGCOMMENT. Symmetrically, LONGCOMMENT has no effect if it is within the scope of IGNORE.

```
  ignore
  statements/text etc. ignored
  ignore  off
```

The format of IGNORE OFF is another exception to the general rule that VPLX commands may be continued on multiple lines. The "OFF" must appear on the same line as "IGNORE."

IGNORE has the same refinements as LONGCOMMENT: 1) it is ended by an end of file on an INCLUDEd file, and 2) its effect persists until the number of IGNORE OFF statements balances the IGNORE statements that have been encountered. For example, two IGNORE statements must be balanced by two IGNORE OFF statements before VPLX resumes interpreting other statements from the command file (*5*).

13.9 ECHO

ECHO controls printing of commands to the print file. The forms are:

- ECHO OFF - to eliminate almost all normal messages and echoing of commands to the print file.
- ECHO or ECHO ON - to resume messages and echoing of commands to the print file.
- ECHO LITE - to allow selective printing of messages and echoing of initial commands of steps, substitutions based on SET (Section 13.10), and INCLUDE statements. Thus, this option is intermediate in degree between the preceding two. Thus, the print file shows an outline of what was done, without the details.

Regardless of the current status of ECHO, the ECHO statements themselves and fatal error messages are sent to the print file.

ECHO affects printing only. ECHO has no effect on which commands are executed.

Unlike LONGCOMMENT and IGNORE, for which levels are counted and whose effect ends once the end of file is reached, each ECHO command takes effect immediately and persists across different levels of INCLUDE

As an illustration, the 2-line command file:

```
echo  lite
include  exam8.crd
```

results in the print file:

```
echo  lite

include exam8.crd


create  in = example7.dat  out = example7.vpl

    Generalized replication assumed

    Size of block   1  =             3

    Total size of tally matrix =     3

    I/O error on primary input file after obs #       6

transform  in = example7.vpl out=exampl7a.vpl

    Unnamed scratch file opened on unit 13
REPLICATE  0, V1=   36.00, V2=   24.00 RATIO=  1.5000
REPLICATE  1, V1=   36.00, V2=   26.00 RATIO=  1.3846
REPLICATE  2, V1=   34.00, V2=   20.00 RATIO=  1.7000
REPLICATE  3, V1=   38.00, V2=   24.00 RATIO=  1.5833
REPLICATE  4, V1=   36.00, V2=   26.00 RATIO=  1.3846


display  in = exampl7a.vpl

                                                   Estimate      Standard error

Number of rooms         :  MEAN                      6.0000               .2357

Persons                 :  MEAN                      4.0000               .4082

Number of rooms         :  TOTAL                    36.0000              1.4142

Persons                 :  TOTAL                    24.0000              2.4495

Rooms per person        :  VALUE                     1.5000               .1356
```

Exhibit 13.2 Print file when EXAM8.CRD is run with ECHO LITE.

The print file contains the INCLUDE statement, each step name, and informational messages from VPLX, as well as the printing results from the TRANSFORM and DISPLAY steps.

The 2-line command file:

```
echo  off
include  exam8.crd
```

results in the print file:

```
echo  off


    I/O error on primary input file after obs #       6
REPLICATE  0, V1=   36.00, V2=   24.00 RATIO=  1.5000
REPLICATE  1, V1=   36.00, V2=   26.00 RATIO=  1.3846
REPLICATE  2, V1=   34.00, V2=   20.00 RATIO=  1.7000
REPLICATE  3, V1=   38.00, V2=   24.00 RATIO=  1.5833
REPLICATE  4, V1=   36.00, V2=   26.00 RATIO=  1.3846
                                                   Estimate      Standard error


Number of rooms         :  MEAN                     6.0000               .2357

Persons                 :  MEAN                     4.0000               .4082

Number of rooms         :  TOTAL                   36.0000              1.4142

Persons                 :  TOTAL                   24.0000              2.4495

Rooms per person        :  VALUE                    1.5000               .1356
```

Exhibit 13.3 Print file when EXAM8.CRD is run with ECHO OFF.

The print file now contains only the warning message from the CREATE step and the printed results from the TRANSFORM and DISPLAY steps.

13.10 SET and Substitution Strings

13.10.1 General Description. The SET command may be used at any point to assign a symbolic name to a string of characters. The syntax is:

SET *stringname* = *set of characters*

Once `stringname` has been defined in this way, then VPLX will substitute the assigned set of characters in place of `&stringname&` on all subsequent occurrences.

For example, one might assign:

```
  set  basedatafile = /data5/survey.dat
  set  workdirect  = /work/
```

The subsequent appearance in the input command file of

```
  create    in = &basedatafile&
            out = &workdirect&tally1.vpl
```

will appear in the VPLX print file as:

```
  create    in = &basedatafile&
  create    in = /data5/survey.dat

            out = &workdirect&tally1.vpl
            out = /work/tally1.vpl
```

to show both the input form and the resulting substitutions. (Line spacing is used in the listing, however, to emphasize the pairing of input line with resulting substitution.) VPLX acts on the statements after the resulting substitutions.

Like other statements in Sections 13.5 - 13.12, SET statements may appear anywhere. A subsequent SET statement can reassign a new string to a previously assigned string name:

```
  set   mmmyy = jan91
  include  monthtal.vsk
  set   mmmyy = feb91
  include  monthtal.vsk
  set   mmmyy = mar91
  include  monthtal.vsk
```

where `montht al.vsk` is a file employing `&mmmyy&` to define files or other information to process one month's worth of data.

The SET statement facilitates the design of general VPLX applications. Suppose, for example, that one wanted to run the same VPLX application on two different computer platforms. A VPLX program could begin with SET statements to define each of the files or directories used by the application, and all subsequent referencing of files could be through the assigned string names. Simply by changing the SET statements at the beginning, one could then use the application on a different data set or port the application to a different computer environment.

13.10.2 Apostrophes. Normally, the length of the string is determined by the first nonblank character after the " = " in the SET statement, and the last nonblank character. To begin a string with one or more blanks, a single apostrophe must be used. In general, an initial apostrophe is never considered part of the string. Similarly, an ending apostrophe may be used to signal the end of the string, and is not treated as part of the string. Consequently, the next two lines from the input file:

```
  set  basedatafile = '' node01"name passwd"::survey.dat  ''
  create   in = &basedatafile&
```

will be displayed on the VPLX listing as

```
  set  basedatafile =  ''node01"name passwd"::survey.dat  ''

  create   in = &basedatafile&
  create   in =  'node01"name passwd"::survey.dat'
```

which removes the outer set of apostrophes. (Recall from Section 13.2.2 that the file name, which includes a blank, must be enclosed in apostrophes.) Note again that the print file shows both the input command and the resulting substitutions.

13.10.3 Rules on Substitution. Except for the range of IGNORE, Section 13.8, and MACROWRITE, Section 13.11, VPLX will treat &*stringname*&, that is, an initial "&", a valid VPLX name (following the same rules as variable names), and an ending "&", as a request for a substitution based on a previously defined SET statement. If no match exists, VPLX treats the occurrence as a fatal error, unless it is in the range of COMMENT, LONGCOMMENT.

As it makes substitutions, VPLX will allow the incoming line to expand into a buffer of 240 characters, which it will then interpret. For example,

```
  reweight in = &datafile& out = &newrepwts& vplxin = &transout&
```

could easily be longer than 80 characters after substitution. As long as the substitutions do not result in a line of greater than 240 characters, VPLX will interpret it. A fatal error will occur if the expansion exceeds this limit, however.

Concatenation of strings is allowed:

```
  set   workdirect = /work/
  set   tallyfile = tally1.dat
  display   out = &workdirect&&tallyfile&
```

appears in the VPLX listing as:

```
set   workdirect = /work/
set   tallyfile = tally1.dat

display   out = &workdirect&&tallyfile&
display   out = /work/tally1.dat
```

If, however, VPLX encounters a " && " in which the first ampersand is not the end of a previous `&stringname` & as above, then VPLX will include a single " & " on the print file and not attempt a substitution. For example,

```
comment  &&tallyfile&& will be defined later
```

appears in the print file as

```
comment  &tallyfile& will be defined later
```

without any attempt to substitute a string, whether or not `&tallyfile&` has been defined in a previous SET statement at that point.

Substitutions may occur within SET statements:

```
set  mmmyy  = jan91
set  outfile = &mmmyy&cps1.dat
display  out = &outfile&
```

which results in:

```
set  mmmyy  = jan91

set  outfile = &mmmyy&cps1.dat
set  outfile = jan91cps1.dat

display  out = &outfile&
display  out = jan91cps1.dat
```

13.10.4 Multiline Substitutions. A string name may be set to more than one line,

```
set newrepwts  = temp$:repweights.dat
     (lrecl = 1100)
reweight  in = datafile.dat
          out = &newrepwts&
```

results in the interpretation:

```
set newrepwts  = temp$:repweights.dat
     (lrecl = 1100)

reweight  in = datafile.dat

          out = &newrepwts&
          out = temp$:repweights.dat
 (lrecl = 1100)
```

In this example, VPLX added a blank before `(lrecl = 1100)` . Note that VPLX generally assumes that any continuation line of a SET statement is intended to be used as a continuation line and will place an initial blank as the second and subsequent lines. An initial apostrophe may be used to direct VPLX to do otherwise, however, as in:

```
set  inputformat = input  x1 x2 weight repw1 - repw100
     'format  (2f3.0,101f10.7)'
```

Generally, multiline SET statements are primarily intended to accommodate extensive file information. In general, however, SET is not well suited for representing multiple lines of VPLX instructions, in spite of the preceding example. MACROWRITE in the next section is a preferable means to represent multiple lines of VPLX code.

13.11 MACROWRITE

If VPLX encounters,

```
MACROWRITE   filename
one or more lines of information
MACROEND
```

it will single-mindedly begin to write each line from the incoming command file to ***filename***. Section 13.13 includes an example. The lines are written out simply as encountered -- INCLUDE, SET, substitution strings, *etc*,. are all simply copied without any other action. This process is ended only by the end of file or by a line with "`MACROEND `" in positions 1-9, in upper or lower case. The specified file then becomes free to be INCLUDEd one or more times later in the run. Alternatively, the output file could be a short data file.

The term "MACROWRITE" suggests its primary use to write out the closest equivalent of "macros" in the VPLX language, which would be a file containing one or more lines of VPLX code with substitution strings, `&stringname&` , to be defined outside of the macro. Of course, such files could be prepared externally with an editor instead. The advantage of MACROWRITE is one of internal documentation: if MACROWRITE is used to write out all of the files that are to be INCLUDEd during the run, it is often easier to develop a VPLX application.

13.12 OFF and ON

The command OFF is a synonym for IGNORE. The line OFF OFF, is equivalent to IGNORE OFF.

ON is not the direct opposite of IGNORE, that is, it does not stand for IGNORE OFF. Instead, it is effectively a 1-line COMMENT. Similarly, ON OFF also has the same force as a comment, rather than having the meaning IGNORE. If ON appears within the scope of a previous OFF or IGNORE, it does not change the effect of the OFF or IGNORE.

The asymmetric meaning of OFF and ON serves a purpose. When used with SET, OFF and ON provide a simple means to identify blocks of code that can be turned off or on by setting a SET statement. To illustrate:

```
set  totalvar = on
set  withinvar = off
...
&totalvar&
one or more lines of VPLX code to be executed only for
   computing total variance (unconditional)
&totalvar&  off
&withinvar&
one or more lines of VPLX code to be executed only for
   computing within variance (conditional on the selection
   of primary sampling units)
&withinvar&  off
```

would execute the lines of code for total variance but not within variance. When VPLX sees `&totalvar&,` it translates it to `on,` simply ignores it as a comment, and continures to process the instructions for total variance. When it gets to `&totalvar&  off,` the translation to `on off` is similarly treated as a 1-line comment. On the other hand, `&withinvar&` and `&withinvar& off` have the same effect as `ignore` and `ignore  off,` causing VPLX to ignore instructions for within variance. Simply by changing the two SET statements at the top, however, one could compute within variance instead.

13.13 An Example

The following example file, newex40.crd, illustrates several of the features in Sections 13.5-13.12.

```
comment  newex40
macrowrite  test.vsk
echo  &echolevel&
comment  EXAM11
comment  This example starts from the data similar to EXAM4 but adds
         an additional variable TENURE.
create  in = &datafile&  out = exampl11.vpl
input    rooms persons cluster tenure / option nprint = 3
format   (4f2.0)
comment  The input data set contains
 5 7 1 2
 6 8 2 2
 5 2 3 1
 4 1 4 2
 8 4 5 1
 8 2 6 1
comment  Create an individual-level ratio of rooms to persons
divide  rooms by persons into proom_indiv
comment  For purposes of illustration, use the IF construction to
     obtain totals for renters and owners.
if  tenure (2) then
_   copy rooms persons into rooms_rent persons_rent
_   constant 1 into rent_count
_   comment  print cases where the number of rooms is 2.0 or more times
       number of persons, for renters only
_   if  proom_indiv (2 - high ) then
_       print  rooms persons cluster
_   end if
else  if tenure (1,3) then
_   copy rooms persons into rooms_own persons_own
_   constant 1 into own_count
end if
comment  unconditionally print variables to here.
print  rooms persons cluster tenure proom_indiv
       rooms_rent persons_rent rent_count
       rooms_own persons_own own_count
longcomment
this line will be printed but not executed
longcomment off
ignore
this line will be neither executed nor printed
ignore  off
&displayopt&
display
option   ndecimal=2 totals
list    rooms      persons
        rooms_rent persons_rent rent_count
        rooms_own  persons_own  own_count
&displayopt&  off
macroend
set  datafile = exampl11.dat
set  echolevel = on
set  displayopt = on
include  test.vsk
set  echolevel = lite
set  displayopt = off
include  test.vsk
set  echolevel = off
include  test.vsk
delete   test.vsk
```

Exhibit 13.4 Example newex40.crd, modified from exam11.crd.

MACROWRITE initially writes most of the lines to `test.vsk`. This file is then INCLUDEd 3 times, with different settings of ECHO and `displayopt`. The first time through produces the largest amount of displayed information. The DISPLAY step is turned off during the 2nd and 3rd time through, which vary by the settings for ECHO. At the very end, the file is DELETEd (Chapter XX). The print file from the run has not been offered here, but intrepid readers can try running this file to inspect the outcome.

NOTES

1. It is helpful to maintain a distinction between the statements described in this chapter and some short or single line steps such as DELETE and FREE, described in Chapter XX. DELETE and FREE constitute steps because VPLX considers the specification of any step to end when DELETE or FREE follows.
2. The alert reader may note that, based on the description thus far in the documentation, file names would be limited to 79 characters, since the first position of a continuation line cannot be a letter or underscore, " _ ". In fact, 80 characters can be achieved through the SET feature described in Section 13.10. SET offers a number of ways to represent a file name of 80 characters. For example, the file name can be broken into 2 parts, and each represented by a string name, as in:

   set directory = *longname1*
   set primarydata = *longname2*
   CREATE   in = &directory&&primarydata&

3. A version using SET:

```
set   finalrepwts =    rewt.dat
                          ...
reweight   in =  testdata1.dat
           out = &finalrepwts&
```

   Generally, SET is better suited than INCLUDE to the specification of individual files.
4. If VPLX encounters more LONGCOMMENT OFF statements than required to balance a previous LONGCOMMENT, it prints a warning message but continues.
5. As with LONGCOMMENT, VPLX will simply print a warning message if it encounters more IGNORE OFF statements than needed to balance previous IGNORE statements.